AD-A082 850   CARNEGIE-MELLON UNIV PITTSBURGH PA MANAGEMENT SCIENC--ETC   F/G 12/1
A STUDY OF THE BOTTLENECK SINGLE SOURCE TRANSPORTATION PROBLEM.(U)
MAR 80   R V NAGELHOUT, G L THOMPSON   N00014-75-C-0621
UNCLASSIFIED   MSRR-456   NL

1 OF 1
AD A
082850

END
DATE
FILMED
5-80
DTIC

See 1473 (13)

B.S. LEVEL II

ADA082850

Carnegie-Mellon University

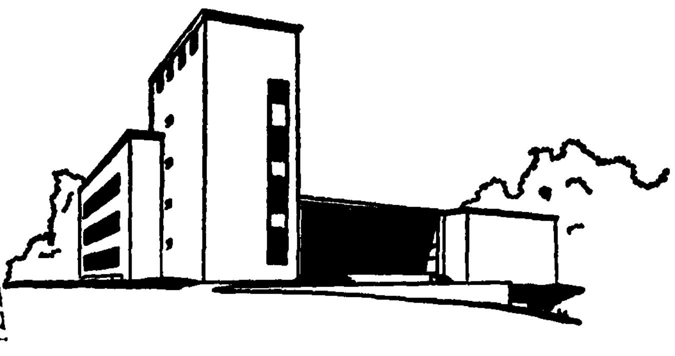
PITTSBURGH, PENNSYLVANIA 15213

APR 9 1980

A

GRADUATE SCHOOL OF INDUSTRIAL ADMINISTRATION

WILLIAM LARIMER MELLON, FOUNDER



DDC FILE COPY

80 4 7 003

✓W.P. #54-79-80

Management Sciences Research Report No. 456

A STUDY OF THE BOTTLENECK SINGLE SOURCE TRANSPORTATION PROBLEM

by

Robert V. Nagelhout

and

Gerald L. Thompson

March 1980

This report was prepared as part of the activities of the Management Sciences Research Group, Carnegie-Mellon University, under Contract N00014-75-C-0621 ✓ NR 047-048 with the Office of Naval Research. Reproduction in whole or in part is permitted for any purpose of the U.S. Government

Management Sciences Research Group ✓
Graduate School of Industrial Administration
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

A STUDY OF THE BOTTLENECK SINGLE SOURCE TRANSPORTATION PROBLEM

by

Robert V. Nagelhout and Gerald L. Thompson

ABSTRACT

Given a set of users with known demands, a set of suppliers with known supplies, and known costs of shipping between suppliers and users, the Bottleneck Single Source Transportation is that of assigning the users to the suppliers so that the following conditions are satisfied: (i) the demand of each user is satisfied by a single supplier; (ii) the amount supplied by each supplier does not exceed its capacity; (iii) the maximum cost of supplying any user by its unique supplier is minimal.

Applications of this problem include: voter redistricting, shipping perishable goods, assignment of city blocks to emergency facilities, and others.

We first discuss a heuristic method which tries to find a feasible solution by assigning the users in order of decreasing demand to the suppliers according to certain orders. In our experience on randomly generated problems, the heuristic was able to find a good, and frequently an optimal feasible solution, most of the time.

If the heuristic fails to find a solution, or finds a solution which is not known to be optimal, then a branch and bound algorithm, using ordinary bottleneck transportation problems as relaxations, is employed to find the optimum single source solution.

Computational experience on some randomly generated problems up to 100 x 400 is presented which indicates that these problems can be solved in less than

2 minutes. Also several small problems with real data were solved; in all but one of these problems the heuristic succeeded in finding an optimal solution.

Key Words

Bottleneck single source
Bottleneck transportation
Integer programming

1. Problem Formulation and Applications

Given a set of users, $J = \{1,\dots,n\}$, with known demands, d_j, and a set of suppliers, $I = \{1,\dots,m\}$, with known supplies, s_i, let c_{ij} be the cost of having the demand of user j satisfied by supplier i. Then the Bottleneck Single Source Transportation Problem (BSSTP) is that of assigning the users to the suppliers so that (i) the demand at each user point is satisfied entirely by a single supplier, (ii) the amount supplied by each supplier does not exceed its capacity, and (iii) the maximum cost of supplying a user by its unique supplier is minimized. Letting x_{ij} denote the quantity shipped from supplier i to user j, we can formulate the BSSTP as:

$$\text{BSSTP} \left\{ \begin{array}{llr} \text{Minimize } \max\limits_{x_{ij} > 0} \{c_{ij}\} & & (1) \\ \text{Subject to} & & \\ \sum\limits_{j \in J} x_{ij} \le s_i & i \in I & (2) \\ \sum\limits_{i \in I} x_{ij} \ge d_j & j \in J & (3) \\ x_{ij} = 0 \text{ or } d_j & i \in I,\ j \in J & (4) \end{array} \right.$$

In this mathematical formulation constraints (3) and (4) ensure adherence to criterion (i), constraints (2) to criterion (ii), and constraint (1) to criterion (iii). The constraints (2) - (4) are those of a Single Source Transportation Problem (SSTP) [12], [15], and they are a special case of the constraints which comprise the Generalized Assignment Problem (GAP) [9], [10], [13], [14]. The objective function in the SSTP and the GAP is that of minimizing total cost whereas the objective in the BSSTP is to minimize the bottleneck cost. In many cases the bottleneck objective is the more relevant criterion. For example consider a voter redistricting problem where I represents a set of voting locations, J is a set of geographic areas or blocks; s_i equals the number of voters who can be accommodated in one day by voting place i; b_j equals the expected number of voters in block j; and c_{ij} equals the time required for a voter in block j to travel to location i. Then the problem is to assign the voting blocks to the locations so that the maximum time for any voter to travel to his (her) voting place is minimized.

The requirement that each user's demand be satisfied from a single source point is also very common in physical distribution problems. For example in the shipment of perishable commodities such as poultry, milk, and other dairy products from distribution centers to local demand centers it is often required that the entire shipment to a local center be made from a single distribution center, and due to costs, such as spoilage, theft, and insurance, it is desirable to minimize the longest time required for shipment of the goods.

Ross and Soland [14] and Klastorin [9] have demonstrated that many location problems can also be viewed as a GAP or a SSTP. Similarly we can formulate bottleneck location problems in the framework of a BSSTP.

For example, consider the voter redistricting problem mentioned previously. Suppose that there is a restriction on the number of voter locations which may be opened; for instance suppose that at most k locations can be opened. Let S be the maximum number of voters which can be handled by any location in one day. Then the voter redistricting location problem would be to choose at most k voter locations out of the set, I, of potential sites, and to assign the groups of voters to these k locations so that the maximum time required for a voter to travel to his (her) voting site is minimized. Figure 1 shows how this location problem can be modelled as a BSSTP. There are m potential voting sites and n groups of voters, where each group lives on a given residential block, which must be assigned to the voting sites which are opened. When $1 \leq j \leq n$, the entries in column j of the matrix give the travel times, c_{ij}, from residential block j to voting site i, for $1 \leq j \leq n$; the entry d_j gives the expected voter turnout from block j. When $n+1 \leq j \leq n+m$, there are exactly two zero entries in column j, $c_{i,n+1} = 0$ and $c_{m+1,n+1} = 0$. All other entries in column j are set equal to infinity, so that they will not be used in any optimal solution. The entries in column $m+n+1$ are all 0's, corresponding to the cost of unused capacity at the voter sites.

Consider now a single source solution to the problem shown in Figure 1. In column $n+i$, where $1 \leq i \leq m$, there are only two possibilities:

$$x_{i,n+i} = 0 \quad \text{and} \quad x_{m+1,n+i} = S \tag{5}$$

or

$$x_{i,n+i} = S \quad \text{and} \quad x_{m+1,n+i} = 0 \quad . \tag{6}$$

It is easy to see that a solution satisfying (5) means that voting site i is open, while a solution satisfying (6) means that voting site i is closed.

In Section 2 we describe a heuristic method which provides an upper bound on the value of an optimal solution to the BSSTP. In Section 3 we describe an algorithm for solving the BSSTP which is utilized whenever the heuristic fails to find an optimal solution. A small example is provided. In Section 4 we discuss our computational experience on some randomly generated problems, and with some problems derived from data taken from the literature. Finally, Section 5 contains some concluding remarks.

2. A Heuristic for BSSTP.

In this section we present a heuristic method for finding a feasible solution to the BSSTP. The heuristic is used to generate a low cost feasible solution to the BSSTP. In particular, in Section 3, we show how to apply the heuristic so that if a feasible solution to the BSSTP is found, it is guaranteed to be optimal.

For notational convenience we define

v = an upper bound on the cost of any positive variable, and

$V_j = \{k | c_{kj} \leq v \text{ and } c_{kj} < \infty\}$

where V_j is the set of all indices of sources whose cost in column j is less than or equal to v. For expositional simplicity we also assume that the demands, d_j, are in nonascending order.

The heuristic begins by randomly assigning the first user demand, d_1, to one of the supply points in V_1. Certain sources with indices in V_1 can be given higher priority over others if this is desirable. For example one may wish to assign d_1 to a source $i \varepsilon V_1$ having the largest supply, s_i, or having the smallest cost c_{i1}. A random choice among the possible supply points is preferred here, however, over a strict rule which always chooses a unique source, since with a random choice rule the heuristic can be repeated several times, generating several feasible solutions, from which the best can be chosen. A "good" choice for v depends upon the problem data (i.e., s_i, d_j, c_{ij}) and we will discuss this choice in more detail later. Once d_1 has been assigned to a randomly chosen source, say k of V_1, then we replace s_k by $s_k - d_1$ which is the amount of supply remaining at source k. Also, for all d_j, $j \geq 2$, such that $d_j > s_k$ we set $c_{kj} = \infty$ signifying that source k can no longer service user j. The assignment of user demands to sources continues sequentially in the same fashion with $d_2, V_2, \ldots, d_n, V_n$, until either, (i) every user demand has been assigned to a source, or (ii) a column, say ℓ, is reached where $V_\ell = \phi$. In case (i) we have a feasible solution to the BSSTP which yields an upper bound, Z^u, on the value of any optimal solution to the BSSTP. In case (ii) the heuristic failed to find a feasible solution and we set the upper bound $Z^u = \infty$. In either case we rerun the heuristic several times in the hope that the random element in the choice of sources will succeed in generating a solution having a small upper bound.

Note that it is possible after several runs, for the heuristic to fail to find any feasible solution to the BSSTP. This is not surprising since the problem of finding a feasible solution to a BSSTP is equivalent to finding a partition, $S_1, \ldots, S_m$, of a set of integers $\{d_1, \ldots, d_n\}$ (the demands) such that the sum of the elements in each S_i is less than or equal to s_i (the supply), for $i = 1, \ldots, m$. The latter problem, which is called the bin packing problem when all the s_i's are equal, is known to be NP complete [4]. Thus it is unlikely that a polynomially bounded algorithm can be found which will be guaranteed to find a feasible solution to the BSSTP. However, in spite of these worst case observations, for the problems which we have tested (See Section 4) we found that a few runs of the heuristic is usually sufficient to generate a low cost feasible solution to a BSSTP.

The only parameter which must be specified to use the heuristic is v. Clearly there is a tradeoff involved in choosing v. If v is too small, then the heuristic may fail to find a feasible solution whereas if v is too large, then the solution value obtained by the heuristic may be so large that it is of no help in reducing the size of the search tree (to be described in Section 3). For the problems which we considered in Section 4 we describe a few of the different parameter settings which we tested. With one of the settings the heuristic either gneerates an optimal solution or it generates no solution at all. We will discuss how and why this is done in Section 3, where an algorithm for solving the BSSTP is given.

Now we formally state the heuristic.

Heuristic 2.1

Step 1. (Initialize). Order the demands so that $d_1 \geq \ldots \geq d_n$. Let $h = 1$. Let $x_{ij} = 0 \quad i\varepsilon I,\ j\varepsilon J$.

Step 2. (Random Assignment) Given v calculate V_h. If $V_h = \phi$ go to (3). Otherwise randomly choose $k\varepsilon V_h$. Let $x_{kh} = d_h$. Replace s_k by $s_k - d_h$. For all $j>h$ if $d_j>s_k$ then let $c_{kj} = \infty$. If $h = n$ go to (4). Otherwise replace h by $h+1$ and go to (2).

Step 3. The heuristic failed to find a feasible solution. Let $Z^u = \infty$.

Step 4. Let $Z^u = \max_{x_{ij}>0}\{c_{ij}\}$. Then Z^u is a valid upper bound for the BSSTP.

Example 1: Let v_{kj} denote the value of the kth smallest cost in column j. In the example we set $v = \max_{j\varepsilon J}\{v_{2j}\}$; that is v equals the maximum, over all columns, of the second smallest cost in each column. The steps of the heuristic on the example in Figure 2 are shown below. The elements of V_j are marked with an *. The chosen variables are circled.

Step	Computation
1	Order demands
2	$v = 8$, $V_1 = \{2,4\}$. Choose $x_{21} = 19$;
2	$v = 8$, $V_2 = \{1,4\}$. Choose $x_{12} = 17$;
2	$v = 8$, $V_3 = \{4\}$. Choose $x_{43} = 17$;
2	$v = 8$, $V_4 = \{3\}$. Choose $x_{34} = 15$
2	$v = 8$, $V_5 = \{1,4\}$. Choose $x_{45} = 10$
2	$v = 8$, $V_6 = \{3\}$. Choose $x_{36} = 8$
4	Stop. $Z^u = 6$.

In practice the heuristic would be repeated several times in hopes of generating many different feasible solutions. Also, the value of the parameter, v, need not remain constant throughout the iterations of Step 2. For example we could set $v = v_{2h}$ at iteration h of Step 2. The value of v at iteration h determines the size of V_h.

In the previous example we would get the following sequence of parameter values and source sets,

Value of h		
1	$v = 4$	$V_1 = \{2,4\}$
2	$v = 7$	$V_2 = \{1,4\}$
3	$v = 9$	$V_3 = \{3,4\}$
4	$v = \infty$	$V_4 = \{3\}$
5	$v = 8$	$V_5 = \{1,4\}$
6	$v = 10$	$V_6 = \{1,3\}$.

The different parameter setting alters V_3 and V_6 and thus with a different random choice of sources it is possible to generate a completely different solution. In the next section we describe an algorithm for solving the BSSTP which is used whenever heuristic 2.1 fails to find an optimal solution.

3. An Algorithm for the BSSTP.

Suppose we relax the constraints (4) in the BSSTP to

$$0 \le x_{ij} \le d_j \qquad i \in I,\ j \in J \tag{7}$$

Then the constraints (1) - (3), (7) are those of a Bottleneck Transportation Problem (BTP). Several algorithms have been proposed and tested for solving a BTP [3], [5], [6], [7], [8], [16], [17]. Let Z_B and Z_B^1 denote the value of an optimal solution to the BSSTP and the BTP respectively. From our computational experience we found that for the overwhelming majority of the BSSTP problems we tested $Z_B = Z_B^1$. That is there was usually no gap between the optimal value of the BSSTP and its relaxation, BTP. This is especially true as the problem size increases. Given this phenomenon, the only heuristic solutions which proved to be of value were those for which $Z^u - Z_B^1$ was "very small," where Z^u is the lowest value obtained by the heuristic. For this reason we tested several different parameter settings for the heuristic 2.1 (see Section 2) in order to generate a low cost solution to the BSSTP. In one of these settings we generate either an optimal solution through the heuristic, or we generate no solution at all. This is done by setting $v = Z_B^1$. Any solution obtained by the heuristic with this parameter setting is guaranteed to be optimal since v is an upper bound on the value of every heuristic solution and Z_B^1 is a lower bound on the value of an optimal solution to the BSSTP.

The drawback to setting $v = Z_B^1$ is that in case $Z_B > Z_B^1$, the heuristic is guaranteed not to find a feasible solution to the BSSTP. However, as we mentioned previously, this occurs so infrequently that it is not of major concern. Also, even when $Z_B > Z_B^1$ it is possible to find a solution to the BSSTP through the heuristic by periodically increasing the value of v. For example we could start with $v = Z_B^1$ and then if after several runs the heuristic fails to find a feasible solution, we could increase v by one and continue in this fashion until a solution is ultimately found.

Remark: In the case where $d_j = 1$ for all $j \in J$ an optimal basic solution to the BTP will also be an optimal solution to the BSSTP. This is a consequence of the fact that any basis matrix for the BTP is unimodular. That is, if the supplies and the demands in the BTP are integer, the solution will also be integer.

In this particular case the only possible integer solution values are $x_{ij} = 0$ or 1 for all $i \varepsilon I$, $j \varepsilon J$, and thus any basic solution to the BTP will satisfy (4).

The algorithm for solving the BSSTP begins by applying the algorithm in [16] to solve the BTP, which provides the absolute lower bound, Z^1_B. If all of the variables in the optimal solution to the BTP satisfy (4), then we are done, since the optimal solution to the BTP is also an optimal solution to the BSSTP. Otherwise we run the heuristic 2.1 which provides an upper bound, Z^u, on Z_B. If $Z^u = Z^1_B$ then we stop because the heuristic solution is optimal. Otherwise, using the Variable Selection Rule (to be described later) we choose a variable, say x_{ij}, which violates the single source constraints (4) (i.e., $0<x_{ij}<d_j$) and we solve the current <u>restricted</u> BTP, BTP^+_{ij}, which results when we force $x_{ij} = d_j$. This is done by setting $c_{kj} = \infty$ for all $k \neq i$ and then reoptimizing the BTP. After having found an optimal solution, X_c, to the current problem BTP^+_{ij}, and its optimal value Z^c, we then apply the Fathoming Test by testing if (i) X_c violates (4) and(ii) $Z^c < Z^u$. If both (i) and (ii) hold true then we use the Variable Selection rule to choose a variable which violates (4) and proceed as previously described. Otherwise if only (ii) holds true we save the new solution, and in any event we fathom the current solution and backtrack. Backtracking is performed by choosing the last variable which was fixed into the basis (i.e. we use a LIFO search rule [12]), say x_{ij}, and fixing it out of the basis. In this case we solve the restricted BTP, BTP^-_{ij}, which results when we set $x_{ij} = 0$. This is done by setting $c_{ij} = \infty$ and reoptimizing the latest restricted problem. We then apply the Fathoming Test and proceed as previously described. The process terminates when either (i) the list of cells which have been fixed in and out is exhausted or (ii) the optimal values of BTP and BSSTP are shown to be equal. Case (ii) occurs quite often as will be shown in Section 4.

We should also point out that in finding an optimal solution to BTP^{+}_{ij} or BTP^{-}_{ij} we do not resolve the restricted BTP from scratch. In the BTP algorithm of Srinivasan and Thompson [15] there are no dual variables associated with the "primal" problem. Thus altering some of the costs in the primal problem by setting them equal to ∞ as previously described does not require a reduced cost adjustment, since there are no reduced costs. An optimal solution to a restricted BTP can be obtained simply by reoptimizing the problem from which it was derived. This normally requires only a few pivots. Therefore only the original BTP is solved completely from scratch and each subsequent restricted BTP requires only a few additional pivots to resolve.

Now we describe the previously mentioned Variable Selection Rule which is used to choose a variable for branching purposes in the branch and bound algorithm. Let j_c denote the smallest column index among all columns $j \epsilon T$ which violate the single source criterion in the current solution. That is, column j_c contains at least two variables which violate (4). Now let i_c be the row index of any variable satisfying,

$$x_{i_c j_c} = \max_{k \epsilon I}\{x_{k j_c}\} \quad . \tag{8}$$

Then $x_{i_c j_c}$ is selected for branching to the $BTP^{+}_{i_c j_c}$ for which $x_{i_c j_c} = d_{j_c}$. Since some variable in column j_c must be equal to d_{j_c} in an optimal solution to the BSSTP, we choose in (8) a variable in column j_c whose value in the current solution to the restricted BTP is closest to d_{j_c}. This Variable Selection Rule is easy to implement and in practice works as well as any of the other selection rules that we tested.

Now we formally state the algorithm for solving the BSSTP. First some notation:

ℓ	=	level of the search tree
LIST(ℓ)	=	contains the variable which is fixed in or out of the basis of the restricted BTP on level ℓ.
Z(ℓ)	=	the optimal objective function value of the restricted BTP on level ℓ.
Q(ℓ)	=	a list of variables whose costs have been set to ∞ on level ℓ.
A	=	a set of columns containing a variable which has been fixed into the basis.
X_c	=	solution to the current restricted BTP.
Z^c	=	optimal objective function value of the current restricted BTP.

Algorithm 3.1 for solving the BSSTP

Step 1. (Initialize) Let $\ell = 1$; $A = \phi$. Solve the BTP. If X_c satisfies equation (4) go to step 7. Otherwise go to step 2.

Step 2. (Heuristic) Run heuristic 2.1 to get Z^u. If $Z^u = Z^c$ then go to step 7. Otherwise let $Z(1) = Z^c$; replace ℓ by $\ell + 1$ and go to step 3.

Step 3. (Branching) Use the Variable Selection Rule (8) to choose $x_{i_c j_c}$. Let $c_{kj_c} = \infty$ for all $k \neq i_c$. Replace A by $A \cup \{j_c\}$. Let Q(ℓ) be the set of all $j \notin A$ such that $d_j > s_{i_c} - \sum_{j \in A} d_j$. Let $c_{i_c j} = \infty$ for all $j \in Q(\ell)$. Let LIST(ℓ) = $(i_c, j_c)^+$.

Step 4. (Fathoming) Reoptimize the current restricted BTP. If $Z^c \geq Z^u$ then go to step 5. Otherwise if X_c satisfies (4) then save X_c; let $Z^u = Z^c$ and go to step 5. Otherwise let $Z(\ell) = Z^c$; replace ℓ by $\ell + 1$, and go to step 3.

Step 5. (Backtracking) If $\text{LIST}(\ell) = (i_c,j_c)^+$ go to step 5a; otherwise go to step 5b.

5a. For all $k \neq i_c$ set c_{kj_c} back to its original value.

For all $j \in Q(\ell)$ set c_{i_cj} back to its original value.

Replace ℓ by $\ell - 1$; replace A by $A - \{j_c\}$; and go to step 6.

5b. Set $c_{i_cj_c}$ back to its original value. Replace ℓ by $\ell - 1$. If $\ell = 1$ and $\text{LIST}(\ell) = (i_c,j_c)^-$ go to step 7. Otherwise go to step 6.

Step 6. If $\text{LIST}(\ell+1) = (i_c,j_c)^-$ then let $(i_c,j_c) = \text{LIST}(\ell)$ and go to step 5b. Otherwise if $Z(\ell) = Z^u$ then go to step 5. Otherwise let $\text{LIST}(\ell+1) = (i_c,j_c)^-$; let $c_{i_cj_c} = \infty$; replace ℓ by $\ell + 1$; go to step 4.

Step 7. Stop. The current saved solution is optimal with value Z^u.

Example 2: Refer to Figures 3(a) - (e) for an illustration of the example. The steps of the algorithm are given below:

1. $\ell = 1$, $A = \phi$. The solution to the BTP is given in Figure 3(a). $Z^c = 4$. X_c violates (4).

2. Assume for exposition purposes that the heuristic could not find a feasible solution (i.e., $Z^u = \infty$). Let $Z(1) = 4$; $\ell = 2$.

3. Select $x_{i_c j_c} = x_{41}$; $A = \{1\}$; $Q(2) = \{2,3,4,5\}$ $\text{LIST}(2) = (4,1)^+$.

4. See Figure 3(b) for the solution to BTP^+_{41}. $Z^c = 8$. X_c satisfies (4). Save X_c, let $Z^u = 8$.

5. $\text{LIST}(2) = (4,1)^+$.

   5a. Restore costs. Let $\ell = 1$. Let $A = \phi$.

6. Let $\text{LIST}(2) = (4,1)^-$; $c_{41} = \infty$; $\ell = 2$.

4. See Figure 3(c) for the solution to BTP^-_{41}. $Z^c = 6$. Let $Z(2) = 6$; $\ell = 3$.

3. Select $x_{i_c j_c} = x_{12}$. $A = \{2\}$, $Q(3) = \{1,3,4\}$

   $\text{LIST}(3) = (1,2)^+$.

4. See Figure 3(d) for the solution to BTP^+_{12}. $X^c = 6$. X_c satisfies (4). Save X_c and let $Z^u = 6$.

5. $\text{LIST}(3) = (1,2)^+$.

   5a. Restore costs. Let $\ell = 2$. Let $A = \phi$.

6. $Z(2) = Z^u = 6$.

5. $\text{LIST}(2) = (4,1)^-$.

   5b. Restore c_{41}. Let $\ell = 1$.

7. Stop. The solution in Figure 3(d) is optimal with value 6.

Figure 3(e) gives a description of the search tree. At node 1 the BTP is solved and has a value of 4. Since the solution to the BTP is not single source (see Figure 3(a)), we use the Variable Selection Rule to choose x_{41}. We then solve BTP^+_{41} at node 2 and find the solution to be single source, with a value of 8. (see Figure 3(b)). Next we solve BTP^-_{41} at node 3 and get a non-single source solution with a value of 6 (see Figure 3(c)). We then use the Variable Selection Rule to choose x_{12} and solve BTP^+_{12}. This yields a single source solution with a value of 6 (see Figure 3(d)). There is no need to generate BTP_{12}^{-} since $Z^u = Z_c = 6$ at node 3.

In this example $6 = Z_B > Z_B^1 = 4$. For all the problems which we report in Section 4, $Z_B = Z_B^1$. Note that in moving from node 3 to node 4, the value of the restricted BTP remained at 6. The solutions at node 3 and 4 are alternate optimal solutions to BTP_{41}^{-}. The solution at node 4 satisfies (4), while the solution at node 3 does not. For a given BTP there are usually an enormous number of alternate optimal solutions. This is probably the reason why $Z_B = Z_B^1$ in most of the problems we tested. That is, among the enormous set of alternate optimal solutions to the BTP there is usually at least one which satisfies (4). In the next section we present an extensive computational study of heuristic 2.1 and the branch and bound algorithm for the BSSTP.

4. Computational Results

In this section we discuss our computational experience on a set of randomly generated problems ranging in size from 10×10 to 100×400, and on a set of problems which were constructed using data from Kuehn and Hamburger [11]. The CPU times in Tables 1 - 3 are subject to some measurement error due to variable loads on the time sharing system.

The random problems were generated in the following manner. We use a uniform probability distribution to generate random integer demands, d_j $j \in J$, between 20 and 200; similarly we generate random integer costs, c_{ij} $i \in I$ $j \in J$, between 1 and 100. Then letting j_1 be the index of a smallest cost entry in column j we set

$$x_{j_1 j} = d_j \quad \text{and} \quad x_{ij} = 0 \quad \text{for} \quad i \neq j_1, \qquad j \in J. \tag{9}$$

We then calculate the largest supply,

$$S = \max_{i\varepsilon I}\{A_i | A_i = \sum_{j\varepsilon J} x_{ij}\} \tag{10}$$

so that if $S_i = S$ for all $i\varepsilon I$, then the solution in (9) will be feasible. Then for each $i\varepsilon I$ we let $s_i = \alpha S(\alpha \leq 1)$, and $d_{n+1} = m\alpha S - \sum_{j\varepsilon J} d_j$ where d_{n+1} is a "dummy" or "slack" demand center.

To have $d_{n+1} > 0$ is a necessary but not sufficient condition for the existence of a feasible solution to the constraints (2) - (4). In general, the smaller the value of d_{n+1}, the more difficult it is to find a feasible solution to a BSSTP. The size of d_{n+1} can be controlled by choosing an appropriate value for α. When $\alpha = 1$ it is easy to see that the solution contained in (9) is optimal. For $\alpha < 1$ the solution in (9) is no longer feasible, and thus the problem is likely to be nontrivial. For the randomly generated problems in Tables 1 - 3 we set α small enough to make the problems as difficult as possible to solve, without making them infeasible. For these problems α varied between .35 and .65.

Table 1 contains the computational results for heuristic 2.1 (see Section 2) on 18 problems ranging in size from 10×10 to 100×400. Three different parameter settings for the heuristic were tested. We first ran the heuristic 40 times setting $v = Z_B^1$ (the optimal value for the BTP). If the heuristic failed to generate an optimal solution then we increased v by ten percent and ran the heuristic over, 40 times. Finally if the second parameter setting failed to generate an optimal solution, the we increased v by ten percent and again ran the heuristic 40 times.

Table 1 gives the iteration out of 40 at which the best heuristic solution is generated, along with the ratio of the heuristic value, Z^u, to the optimal value for the BSSTP, Z_B.

We also calculated for each problem, the number of "fractional" variables in the optimal solution to the BTP. A fractional variable is one which violates (4). We have found that for most single source transportation problems, the computational burden is directly proportional to the number of fractional variables (See [12]). In fact we can easily calculate the upper bound on the number of fractional variables in the following manner. Given an $m \times n$ problem we add one slack column, $n+1$, which has no single source restriction. The total number of basic cells is $m+(n+1) - 1 = m+n$. Each of the $n+1$ columns must contain one basic cell, which leaves $m-1$ basic cells to distribute among $n+1$ columns. The maximum number of fractional cells occurs when $m-1$ of the first n columns contains exactly two fractional cells. In this case there are $2(m-1)$ fractional cells. Note that this upper bound on the number of fractional cells is independent of n. Thus we can increase the number of columns arbitrarily without changing the upper bound on the number of fractional cells.

Table 1 shows that the heuristic 2.1 generated an optimal solution in eleven out of eighteen problems. The heuristic failed to find any feasible solution in five of the problems (7, 10, 11, 14, 15). In problems 7, 10, 11 and 15 the number of fractional variables is greater than that of its counterpart problem of the same dimensions. For example in problem 16, 49 out of a possible $2(100-1) = 198$ variables are fractional whereas for problem 15, 100 out of 198 are fractional. Notice also that although the upper bound on the number of fractional variables does not increase with n, in practice the actual number of fractional variables does increase with n.

Table 2 contains data for those problems in Table 1 which were not solved to optimality by heuristic 2.1. We report the number of nodes, the number of pivots, the number of feasible solutions obtained in the process of locating the optimal, and the additional CPU times required over heuristic 2.1. Again problems 7, 10, and 15, which have a large number of fractional variables, were the most difficult problems to solve. However, none of the problems required an excessive amount of CPU time.

For all of the problems 1 - 18, $Z_B^1 = Z_B$, that is there was no difference between the optimal values for the BTP and the BSSTP. We generated problems with different cost and demand ranges than 1 - 18 and also found that $Z_B = Z_B^1$ for these problems. We found the algorithm to be insensitive to these changes in the data, with the one exception that problems for which d_{n+1} was relatively small tended to be more difficult. Of course it is easy to contrive problems for which $Z_B > Z_B^1$. For example consider any BSSTP in which there are $m \times n$ different values for c_{ij}, $i \epsilon I$ $j \epsilon J$. Then as long as the solution to the BTP is not identical to the solution to the BSSTP we will have $Z_B > Z_B^1$.

We believe that an intuitive explanation for the $Z_B = Z_B^1$ phenomenon is that the BTP has a large number of alternate optimal solutions, among which at least one satisfies (4). It is well known that the BTP tends to have many more alternate optimal solutions than its total cost counterpart, the ordinary transportation problem.

Table 3 contains the results for some problems which were derived from the 24×50 Kuehn and Hamburger data contained in [11]. The c_{ij} values are actual distances from 24 potential warehouse sites to 50 demand centers across the United States. The d_j values represent the population at demand center j. The source values s_i, $i \epsilon I$ were randomly generated as in (9) and (10) using $\alpha = .7$. The demands range from 32 to 12,912 and the costs range from 0 to 3,244. Again for all of the problems in Table 3, $Z_B^1 = Z_B$.

The heuristic generated an optimal solution in nine out of the ten problems and the one problem for which the heuristic failed to find the optimal required only eleven nodes to solve.

5. Concluding Remarks

In this paper we presented a heuristic and a branch and bound algorithm for solving the Bottleneck Single Source Transportation Problem (BSSTP). We showed how location problems can be modelled in the framework of the BSSTP. Next we showed how to specify a parameter in the heruristic so that it either generates an optimal solution to the BSSTP, or it generates no solution at all. Using this method we were able to solve 17 out of the 28 problems in Section 4 without resorting to a branch and bound algorithm.

We found that a good indicator of the difficulty of a BSSTP is the number of fractional variables in an optimal solution to its relaxation, the Bottleneck Transportation Problem (BTP). The larger the number of fractional variables in an optimal solution to the BTP, the more difficult the BSSTP tended to be. We provided an upper bound on the number of fractional variables and showed that although the upper bound is independent of n, the number of columns, the actual number of fractional variables tends to increase when n is increased.

Finally we showed that in those problems for which the heuristic failed to find an optimal solution, the algorithm had little difficulty in locating an optimal solution. The BSSTP algorithm exhibited very little variance in its performance and was capable of solving problems with up to 40,000 integer variables.

$$
\begin{array}{ccc|ccccc|c|c}
1 & & n & n+1 & & & n+m & & n+m+1 & \\
\hline
c_{11} & \cdots & c_{1n} & 0 & \infty & \cdot & \infty & \infty & 0 & S \\
c_{21} & & c_{2n} & \infty & 0 & & \infty & \infty & 0 & \cdot \\
\cdot & & \cdot & \cdot & & \cdot & & \cdot & \cdot & \cdot \\
\cdot & & \cdot & \cdot & & & 0 & \infty & 0 & \cdot \\
c_{m1} & \cdots & c_{mn} & \infty & \infty & \cdot & \infty & 0 & 0 & S \\
\infty & \cdots & \infty & 0 & 0 & \cdots & 0 & 0 & 0 & kS \\
\hline
d_1 & \cdots & d_n & S & \cdot & \cdot & \cdot & S & \uparrow & \\
 & & & & & & & & kS - \sum_{j=1}^{n} d_j &
\end{array}
$$

Figure 1

10	* $\textcircled{6}$ ^{17}	4	8	* 8	10	0 ^{10}	27
* $\textcircled{4}$ ^{19}	2	2	2	8	1	0 ^{7}	26
14	12	9	* $\textcircled{4}$ ^{15}	9	* $\textcircled{3}$ ^{8}	0 ^{3}	26
* 4	* 7	* $\textcircled{6}$ ^{17}	9	* $\textcircled{1}$ ^{10}	4	0	27
19	17	17	15	10	8	20	

Figure 2

BTP

10	6	④^{10}	8	8	10	⓪^{17}	27
④^{2}	②^{17}	②^{7}	2	8	1	0	26
14	12	9	④^{15}	9	③^{8}	⓪^{3}	26
④^{17}	7	6	9	①^{10}	4	0	27
19	17	17	15	10	8	20	

Figure 3(a)

BTP^{+}_{41}

∞	⑥^{17}	4	8	⑧^{10}	10	⓪^{0}	27
∞	2	②^{17}	2	8	1	⓪^{9}	26
∞	12	9	④^{15}	9	③^{8}	⓪^{3}	26
④^{19}	∞	∞	∞	∞	4	⓪^{8}	27
19	17	17	15	10	8	20	

Figure 3(b)

BTP^{-}_{41}

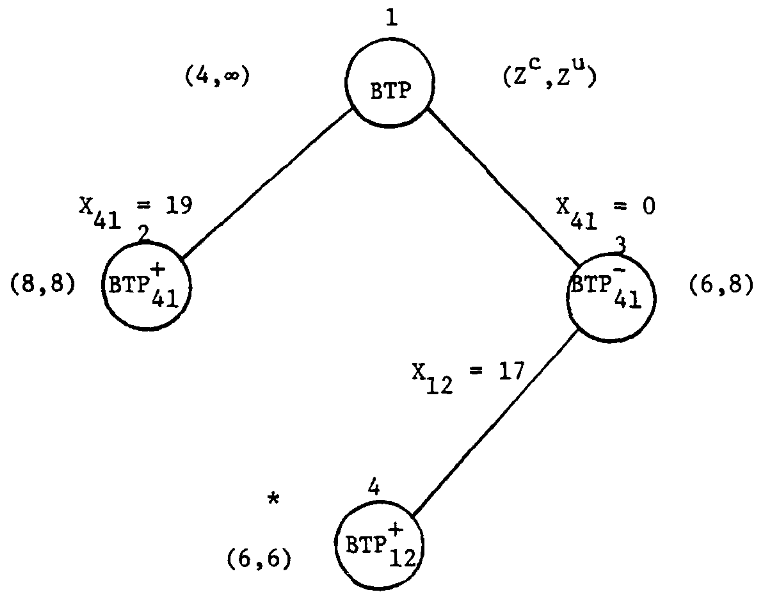
10	$\textcircled{6}^{10}$	$\textcircled{4}^{8}$	8	8	10	$\textcircled{0}^{9}$	27
$\textcircled{4}^{19}$	$\textcircled{2}^{7}$	2	2	8	1	0	26
14	12	9	$\textcircled{4}^{15}$	9	3	$\textcircled{0}^{11}$	26
∞	7	$\textcircled{6}^{9}$	9	$\textcircled{1}^{10}$	$\textcircled{4}^{8}$	0	27
19	17	17	15	10	8	20	

Figure 3(c)

BTP^{+}_{12}

10	$\textcircled{6}^{17}$	∞	∞	8	10	$\textcircled{0}^{10}$	27
$\textcircled{4}^{19}$	∞	2	2	8	1	$\textcircled{0}^{7}$	26
14	∞	9	$\textcircled{4}^{15}$	9	$\textcircled{3}^{8}$	$\textcircled{0}^{3}$	26
∞	∞	$\textcircled{6}^{17}$	9	$\textcircled{1}^{10}$	4	$\textcircled{0}^{0}$	27
19	17	17	15	10	8	20	

Figure 3(d)



1
$(4,\infty)$
BTP
(Z^c,Z^u)
$X_{41} = 19$
2
$(8,8)$
BTP^{+}_{41}
$X_{41} = 0$
3
BTP^{-}_{41}
$(6,8)$
$X_{12} = 17$
*
4
$(6,6)$
BTP^{+}_{12}

Figure 3(e)

Heuristic 2.1

v			Z_B^1		$Z_B^1+.1Z_B^1$		$Z_B^1+.2Z_B^1$		CPU* Time Dec-20 Seconds
No.	Problem Size	No.of Frac.Var.	No.of Iter.	$\frac{Z^u}{Z_B}$	No.of Iter.	$\frac{Z^u}{Z_B}$	No.of Iter.	$\frac{Z^u}{Z_B}$	
1	10×10	2	3	1	-	-	-	-	.08
2	10×10	2	4	1	-	-	-	-	.08
3	10×100	12	40	∞	2	$\frac{49}{46}$	40	-	5.1
4	10×100	15	16	1	-	-	-	-	.85
5	10×200	12	22	1	-	-	-	-	3.7
6	10×200	10	40	∞	18	$\frac{41}{39}$	33	1	15.5
7	50×75	55	40	∞	40	∞	40	∞	7.8
8	50×75	21	7	1	-	-	-	-	.50
9	50×125	36	21	1	-	-	-	-	2.4
10	50×125	54	40	∞	40	∞	40	∞	11.5
11	50×200	64	40	∞	40	∞	40	∞	27.7
12	50×200	56	1	1	-	-	-	-	.6
13	100×100	57	40	∞	1	1	-	-	6.8
14	100×100	53	40	∞	40	∞	40	∞	17.9
15	100×150	100	40	∞	40	∞	40	∞	28.9
16	100×150	49	40	∞	40	∞	24	1	21.0
17	100×400	108	12	1	-	-	-	-	12.48
18	100×400	106	40	∞	40	∞	17	$\frac{7}{6}$	101.7

* Coded in Fortran IV

Table 1

No.	Problem Size	No. of Frac.Var.	No. of Nodes	No. of Pivots	No. of Feasible Solution	DEC-20* CPU Time
3	10×100	12	119	380	2	.7
7	50×75	55	1405	5150	17	10.5
10	50×125	36	6321	18161	16	43.6
11	50×200	64	186	988	5	4.0
14	100×100	53	46	325	5	1.2
15	100×150	100	9263	24879	12	86.5
18	100×400	106	133	609	1	4.3

* Coded in Fortran IV

Table 2

v	Z_B^1		$Z_B + .1Z_B$			
Problem Size	No. of Iterations	$\frac{Z^u}{Z_B}$	No. of Iterations	$\frac{Z^u}{Z_B}$	No. of Nodes	Total CPU* Time DEC-20 Sec.
24×50	7	1	-	-	-	.28
24×50	2	1	-	-	-	.19
24×50	3	1	-	-	-	.19
24×50	8	1	-	-	-	.24
24×50	1	1	-	-	-	.13
24×50	7	1	-	-	-	.30
24×50	20	∞	14	$\frac{667}{656}$	11	1.50
24×50	4	1	1	-	-	.18
24×50	1	1	-	-	-	.15
24×50	7	1	-	-	-	.26

* Coded in Fortran IV

Table 3

References

[1] Demaio, A. and C. Roveda, "An All Zero-One Algorithm for a Class of Transportation Problems," Operations Research, Vol. 19, pp. 1405-1418, (1971).

[2] Derigs, Ulrich and U. Zimmerman, "An Augmenting Path Method for Solving Linear Bottleneck Transportation Problems," Report 77-9, Mathematics Institute, University of Köln, (1977).

[3] Frieze, A. M. "Bottleneck Linear Programming," Operational Research Quarterly, Vol. 26, pp. 871-874, (1975).

[4] Garey, M. R., and D. S. Johnson, Computers and Intractability, W. H. Freeman and Co., San Francisco, 1979.

[5] Garfinkel, R. S., "An Improved Algorithm for the Bottleneck Assignment Problem," Operations Research, Vol. 19, pp. 1747-1751 (1971).

[6] Garfinkel, R. S. and M. R. Rao, "The Bottleneck Transportation Problem," Naval Research Logistics Quarterly, Vol. 18, pp. 465-472 (1971).

[7] Hammer, P. L., "Time Minimizing Transportation Problems," Naval Research Logistics Quarterly, Vol. 16, pp. 345-357, (1969).

[8] Hammer, P. L., "Communication on the Bottleneck Transportation Problem and Some Remarks on the Time Transportation Problem," Naval Research Logistics Quarterly, Vol. 18, pp. 487-490, (1971).

[9] Klastorin, T. D., "On the Maximal Covering Location Problem and the Generalized Assignment Problem," Management Science, Vol. 25, No. 1, pp. 107-116, (1979).

[10] Klastorin, T. D., "An Effective Subgradient Algorithm for the Generalized Assignment Problem," Computers and Operations Research, Vol. 6, pp. 155-164, (1979).

[11] Kuehn, A. A. and M. J. Hamburger, "A Heuristic Program for Locating Warehouses," Management Science, Vol. 9, pp. 643-666, (1963).

[12] Nagelhout, R. V. and G. L. Thompson, "A Single Source Transportation Algorithm" to appear in Computers and Operations Research.

[13] Ross, G. T. and R. M. Soland, "A Branch and Bound Algorithm for the Generalized Assignment Problem" Mathematical Programming, Vol. 8, pp. 91-105 (1975).

[14] Ross, G. T. and R. M. Soland, "Modelling Facility Location Problems as Generalized Assignment Problems," Management Science, Vol. 24, No. 3, pp. 345 - 357, (1977).

[15] Srinivasan, V. and G. L. Thompson, "An Algorithm for Assigning Uses to Sources in a Special Class of Transportation Problems," Operations Research, Vol. 21, pp. 284-295, (1973).

[16] Srinivasan, V. and G. L. Thompson, "Algorithms for Minimizing Total Cost, Bottleneck Time and Bottleneck Shipment in Transportation Problems," Naval Research Logistics Quarterly, Vol. 23, pp. 567-595, (1976).

[17] Szwarc, W., "Some Remarks on the Time Transportation Problem," Naval Research Logistics Quarterly, Vol. 18, pp. 473-485, (1971).

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (14) MSRR-456, WP-54-79-E01	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) A STUDY OF THE BOTTLENECK SINGLE SOURCE TRANSPORTATION PROBLEM.		5. TYPE OF REPORT & PERIOD COVERED Technical Report ~~March 1980~~
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Robert V. Nagelhout, Gerald L. Thompson	(15)	8. CONTRACT OR GRANT NUMBER(s) N00014-75-C-0621
9. PERFORMING ORGANIZATION NAME AND ADDRESS Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR 047-048
11. CONTROLLING OFFICE NAME AND ADDRESS Personnel and Training Research Programs, Office of Naval Research (Code 434), Arlington, VA 22217		12. REPORT DATE (11) Mar 1980
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (12) 32		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

(9) Management sciences research rept,

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Bottleneck single source, Bottleneck transportation, Integer programming.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Given a set of users with known demands, a set of suppliers with known supplies, and known costs of shipping between suppliers and users the Bottleneck Single Source Transportation is that of assigning the users to the suppliers so that the following conditions are satisfied: ~~(i)~~ the demand of each user is satisfied by a single supplier; ~~(ii)~~ the amount supplied by each supplier does not exceed its capacity; ~~(iii)~~ the maximum cost of supplying any user by its unique supplier is minimal. → next page (Over)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

403426 [illegible]

Applications of this problem include: voter redistricting, shipping perishable goods, assignment of city blocks to emergency facilities, and others.

We first discuss a heuristic method which tries to find a feasible solution by assigning the users in order of decreasing demand to the suppliers according to certain orders. In our experience on randomly generated problems, the heuristic was able to find a good, and frequently an optimal feasible solution, most of the time.

If the heuristic fails to find a solution, or finds a solution which is not known to be optimal, then a branch and bound algorithm, using ordinary bottleneck transportation problems as relaxations, is employed to find the optimum single source solution.

Computational experience on some randomly generated problems up to 100 x 400 is presented which indicates that these problems can be solved in less than 2 minutes. Also several small problems with real data were solved; in all but one of these problems the heuristic succeeded in finding an optimal solution.